# Creating a large GUI Program

Dan Fleck

Spring 2007

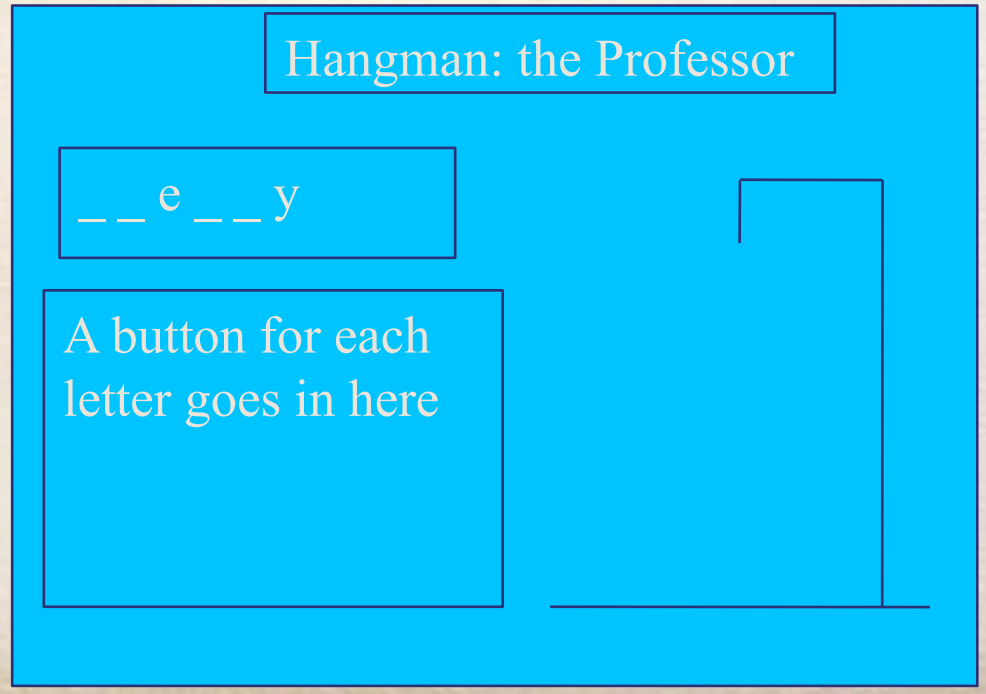
Coming up: Hangman – yes again!

# Hangman – yes again!

- Lets say we want to create a more fun, graphical version of our Hangman game. This is alarger program and we will follow the Software Development Process from Chapter 2
- Steps
  - Analyze the problem
  - Determine the specification
  - Create a design
  - Implementation
  - Testing / Debugging
  - Maintenance

# Hangman Analysis

- Analysis – understand what we want to do. In our simple program, this is a quickly drawn GUI:



# Hangman Analysis

- Analysis – understand what we want to do. In our simple program, this is a quickly drawn GUI
  - Lets make it more fun by using a real picture! So, we'll take a GIF, and cut it into pieces (head, arms, legs…) and use those instead of drawing the hangman. ☺

# Hangman Design

- Design – start thinking about how to do this in a computer. Lets break up our big problem into little problems

- Create a bunch of buttons each with a letter in it
- Create a label that we can display the string in
- Create a canvas with the hangman's holder drawn up, and allow it to a image parts in

# Hangman Design: Button Frame

- Create a bunch of buttons each with a letter in it
  - Pseudocode
    - loop through each letter in the alphabet
      - create a button
      - add the button to a frame
      - bind the button to a function that will determine which button was pressed and then will call another function that handles a specific guess
    - Create the button handler function (described above)

# Hangman Design: Button Frame

- loop through each letter in the alphabet
  - create a button
  - bind the button to a function
  - add the button to a frame

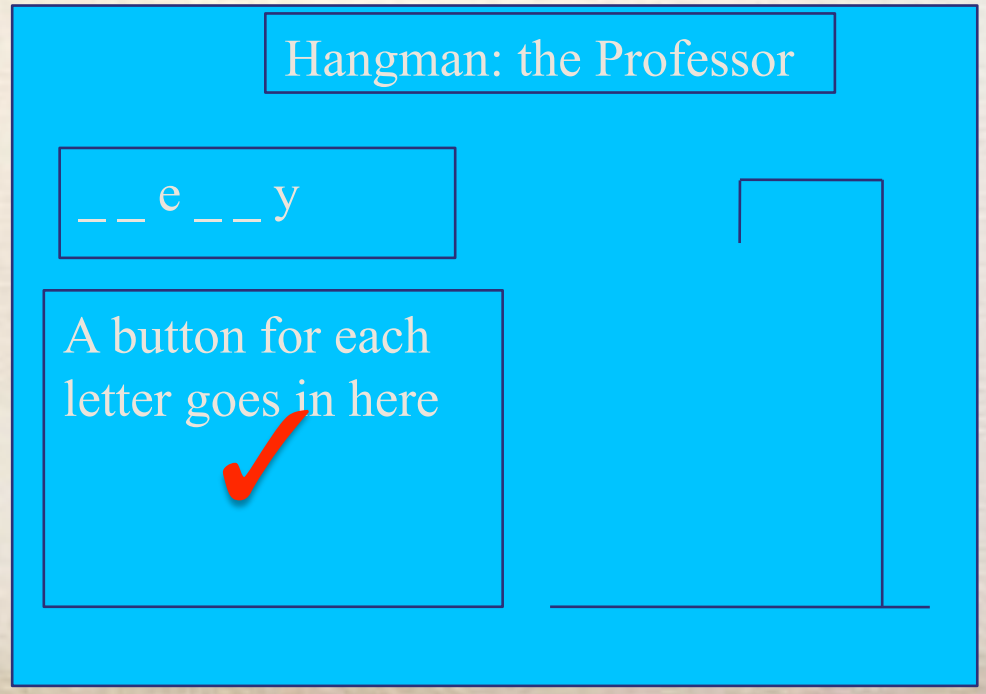
```
def createButtons(parent):
    for i in range(26):
        letter = chr(i) # Convert number to ascii character
        b = button(parent, text=letter,
                                    command=handleButton)
        b.pack()
```

# Button Frame Implementation

- Problems to solve:
  - Want the buttons to be in multiple rows
  - Want the buttons to be grayed out after clicked
  - Want i and j to show up!
  - Want the button handler to know which button was pushed

# Hangman Analysis

- Analysis – understand what we want to do. In our simple program, this is a quickly drawn GUI:



# Hangman: Label Analysis

- Create a label that we can display the string in
  - Make a label with a changeable string
  - Make a function that decides what to show (_ or the letter)
    - Needs to know: lettersGuessed and currentWord
  - Create a function that accepts a guess and maintains all this information

# Hangman: Label Analysis

- Make a function that decides what to show (_ or the letter)
  - Needs to know: lettersGuessed and currentWord

```
loop through all chars in the word
   if charInWord is in lettersGuessed
        show the letter
   else
        show the blank (_)
   *** Need build up a string so we can call
   changeAbleStr.set(stringToShow) ***
```

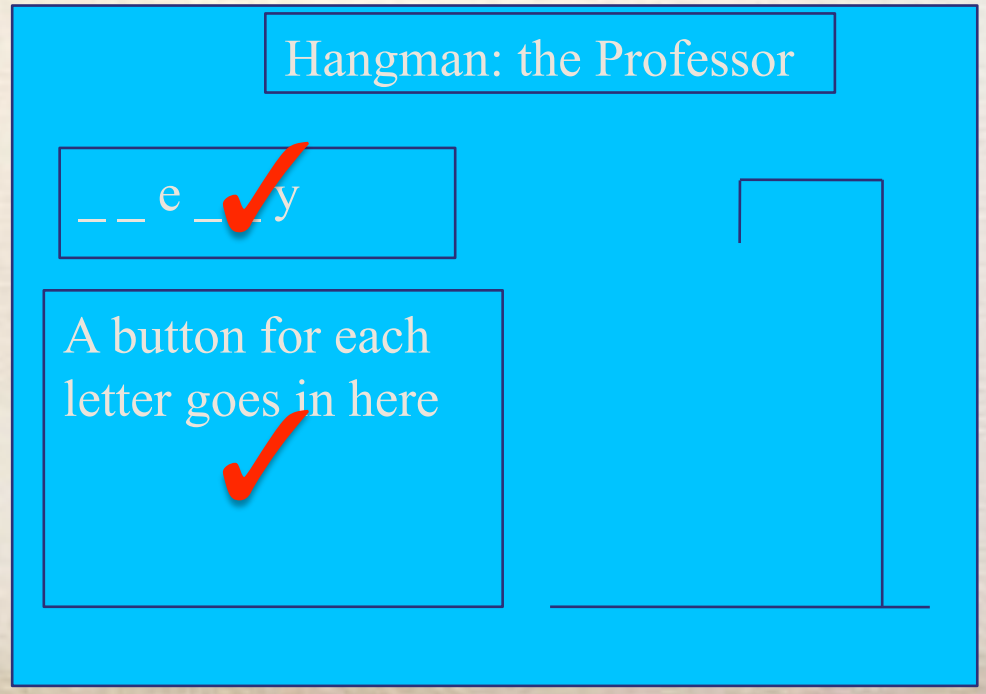
# Hangman: Label Analysis

```
stringToShow = “” # Empty
loop through all chars in the word
   if charInWord is in lettersGuessed
        add letter to stringToShow
   else
        add blank to stringToShow
   changeAbleStr.set(stringToShow)
```

See label1.py

# Hangman Analysis

- Analysis – understand what we want to do. In our simple program, this is a quickly drawn GUI:



# Hangman: Canvas Analysis

- Create a canvas with the hangman's holder drawn up, and allow it to a image parts in

- Steps:
  - Create the canvas
  - Draw the noose
  - Create a function that accepts the body part number, and add it into the canvas
  - Call that function everytime someone misses a guess

# Hangman: Canvas Implemenation

- Create the canvas
  - canvas = Canvas(parent, width=300,height=700)
- Draw the noose
  - canvas.create_line(5,540,220,540) #base
  - canvas.create_line(210, 540, 210, 5) # Long pole
  - canvas.create_line(100, 5, 210, 5)
  - canvas.create_line(100, 5, 100, 10)

# Hangman: Canvas Implementation

- Create a function that accepts the body part number, and add it into the canvas
- How?
  - Option 1: Make a series of pictures each showing more body parts. Then replace the picture with the next picture in the series to add a part
  - Option 2: Make a group of pictures each with one body part. Add the pictures in one at a time

# Hangman: Canvas Implementation

- Option 1: Make a series of pictures each showing more body parts. Then replace the picture with the next picture in the series to add a part

```
images = ['head.gif', 'torso.gif', 'arm1.gif', 'arm2.gif', 'leg1.gif', 'fulldan.gif']
canvas=None # Holds the canvas we're drawing the hangman on
def addPiece(misses):
    global canvas, images
    img = PhotoImage(file=images[misses])
    canvas.create_image(0, 6,anchor=NW, image=img)
    canvas.image = img
```

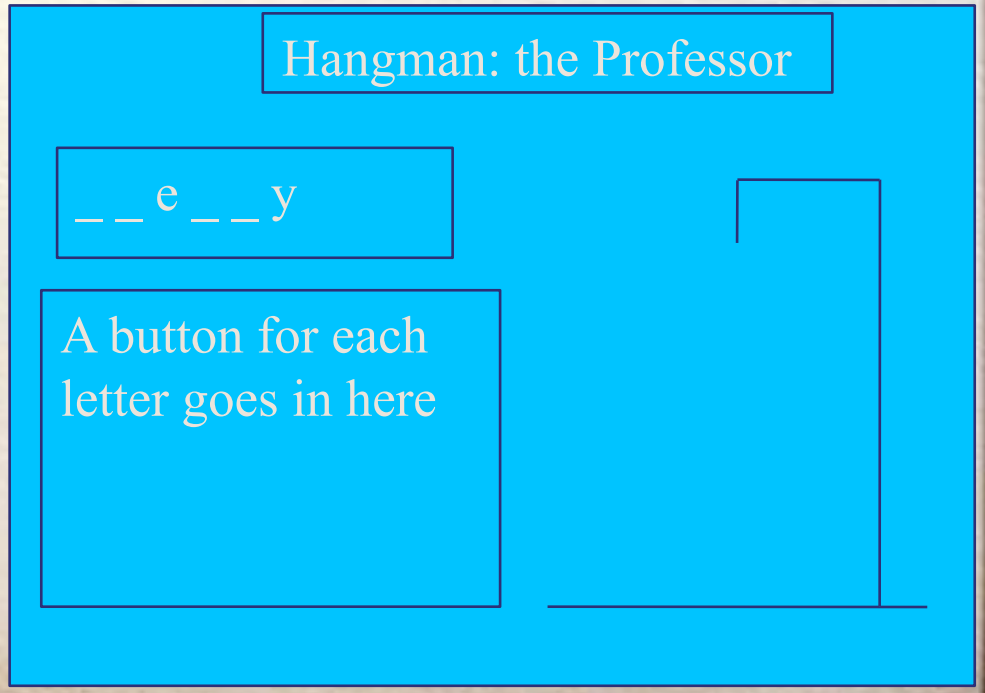
# Things left to do

- Add the title to the top of the window (How?)
- Make a “You lost” message appear
- Make the game use words from a file
- Don’t allow people to click on buttons that are disabled (we used “bind” so they are really still clickable)
- Add sound? ☺

# Recap

- We started with a simple idea and a simple GUI layout

- Analysis
- Design
- Implementation
- Testing/Debug
- Maintenance



# Testing/Debugging

- When testing your goal is to verify all parts of your software work correctly with both good and bad inputs.
- Specifically you want to think about how it *SHOULD* work… not really how you implemented it! (aka Requirements)
- What are some test scenarios you would write for this software?

# Some Test Scenarios

- Validate that solving the puzzle works
- Validate that when you do not solve the puzzle, a message appears and the game ends
- Validate that the software works with words to solve with spaces in them
- Validate that the software works with upper and lowercase words

End of presentation